双输入 7 到 13-Bit 递增 ADC 数据手册 DualADC V 2.2

资源	PSoC® 模块			API 内存 （字节）		引脚 （根据外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27/24xxx, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52						
	4	0	2	444	8	1
CY8C26/25xxx	4	0	2	474	8	1

请参见应用笔记 “Analog - ADC Selection” AN2239 了解其他转换器。

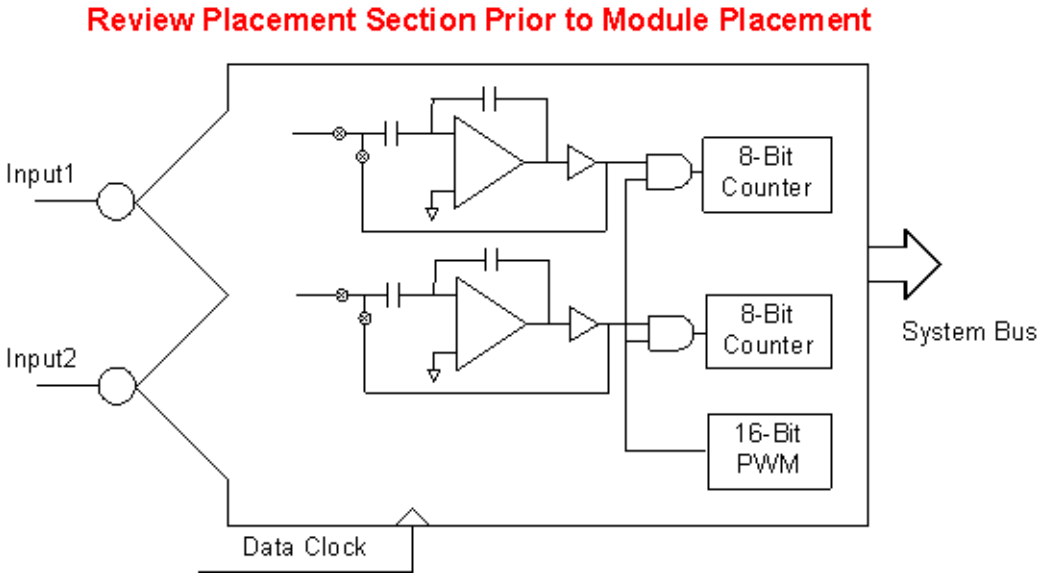
有关使用此用户模块的一个或多个完整配置的实用型示例项目，请转到 www.cypress.com/psocexampleprojects.

功能和概述

- 同时对两个输入进行采样
- 7 到 13-bit 分辨率
- 2 的补码或无符号整数
- 采样率从 4 到大于 10,000 sps
- 多输入范围，包括 Vss 到 Vdd
- 积分转换器提供良好的正常模式抑制
- 内部或外部时钟

DualADC 用户模块是双输入递增 ADC，其分辨率可在 7 到 13 比特之间调整。 它可以配置为通过优化积分时间来删除不需要的高频率。 可以通过配置适当的参考电压和模拟接地来测量输入电压范围 （包括轨至轨）。 输出可以配置为 2 的补码或无符号整数。 DualADC 适合于需要同时对两个信号进行采样的应用场合，如电源测量。 像其他 PSoC ADC 一样，可以复用传递给两个输入的信号。 CPU 负荷因输入电平而异。 例如，当 V_{in} = $+V_{ref}$ 时，有 10,014 个 CPU 周期（最大 13 位）。 当 V_{in} = AGND 时，有 5,278 个 CPU 周期 （平均 13 位）。 当 V_{in} = $-V_{ref}$ 时，有 542 个 CPU 周期 （最低 7-13 位）。

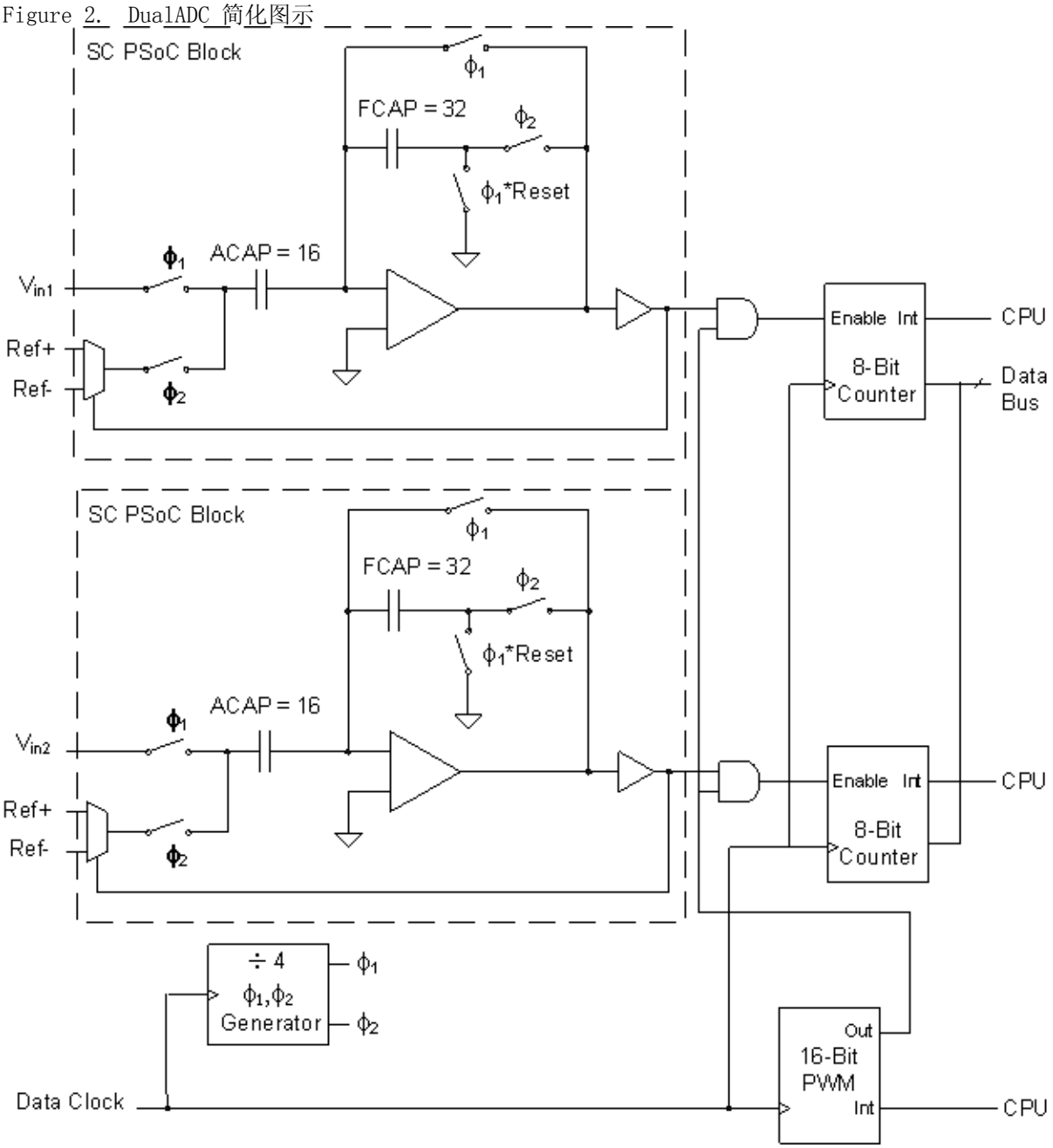
Figure 1. DualADC 模块图



功能说明

DualADC 用户模块由一个用户模块中的两个递增 ADC 组成。 16-bit 采样率定时器 (PWM) 是共享的，这样可以减少所需的数字模块。 由于两个 ADC 都使用同一定时器，采样完全同步。 在转换周期结束时，可以同时获得两个输入的结果。 需要四个数字 PSoC 模块和两个模拟开关电容 PSoC 模块，请参见下面的简化图示。

Figure 2. DualADC 简化图示



两个模拟模块都同样配置为可复位的积分器。 根据输出极性，配置了参考控制，以便参考电压可以加入到输入中或从输入中减去参考电压，然后放入积分器。 此参考电压控制会试图将积分器输出电压拉回至 AGND。 如果积分器运行 2^{Bits} 次，而输出电压比较器在这些次数中结果为正的次数为 “n”，则输出残余电压 (V_{resid}) 如下。

Equation 1

$$V_{resid} = 2^{Bits} \cdot V_{in} - (n \cdot V_{ref}) + (2^{Bits} - n) \cdot V_{ref}$$

Equation 2

$$V_{in} = \frac{n - 2^{Bits-1}}{2^{Bits-1}} V_{ref} + \frac{V_{resid}}{2^{Bits}}$$

此等式说明了此 ADC 的范围为 ± V_{ref}，分辨率 (LSB) 为 $V_{ref}/2^{Bits-1}$，计算结束时输出电压定义为残余电压。 由于 V_{resid} 始终小于 V_{ref}，因此 $V_{resid}/2^{Bits}$ 小于 LSB 的一半，可以忽略。 下面列出了所得到的等式。

Equation 3

$$V_{in} = \frac{n - 2^{Bits-1}}{2^{Bits-1}} V_{ref}$$

示例 1

如果 V_{ref} 为 1.3V 以及分辨率为 8-bits，我们可以根据在数据可用时从递增 ADC 读取的值，轻松计算出输入电压。 可使用的等式如下：

Equation 4

$$V_{in} = \frac{n - 128}{128} 1.3$$

计算结果是相对于 AGND 的电压值。 如果 ADC 数据值为 200，测量电压的计算结果为 0.73V，如下所示：

Equation 5

$$V_{in} = \frac{200 - 128}{128} 1.3 = 0.73V$$

计算出的值为理想值，很有可能因系统噪声和芯片偏移而有所不同。

为了确定在特定输入电压下会出现的代码，可以变换一下等式形式：

Equation 6

$$n = \frac{2^{Bits-1} \cdot V_{in}}{V_{ref}} + 2^{Bits-1}$$

示例 2

如果 V_{ref} 为 1.3V 以及分辨率为 8-bits，我们可以根据输入电压轻松计算出将要出现的 ADC 代码。 可使用的等式如下：

Equation 7

$$n = \frac{128 \cdot V_{in}}{1.3} + 128$$

如果输入电压为 -1V （低于 AGND），根据下面的计算，预计 ADC 代码将为 29.53：

Equation 8

$$n = \frac{128 \cdot (-1)}{1.3} + 128 = 29.53$$

计算出的值为理想值，很有可能因系统噪声和芯片偏移而有所不同。

为了将积分器函数作为递增 ADC，使用了下列数字资源：

- 8-bit 计数器，用于累计输出为正的周期数 （每个通道一个）。
- 16-bit PWM，用于计算积分时间和控制进入 8-bit 计数器的时钟 （在两个通道之间共享）。

单一 DataClock 连接到 8-bit 计数器、16-bit PWM 和模拟列时钟（该时钟连接到模拟 SC PSoC 模块）。模拟列时钟实际上是两个时钟 Φ_1 和 Φ_2，它们是从 DataClock 生成的。 这两个附加时钟是 DataClock 频率的四分之一。 这意味着 PWM 和计数器的运行速度比所需速度快四倍，因此，需要累计 N+2 位数据（N 为分辨率的位数）。

Note 当放置此模块时，必须将它配置为针对所有模块使用相同时钟。 不这样做会导致它运行不正常。

计数器是使用 8-bit 数字模块 （对于 LSB）和软件计数器 （对于 MSB）实现的。 每当硬件计数器溢出时，都会生成中断，计数器的高 MSB 会递增。 这使得 DualADC 模块可以只用四个数字模块而非六个数字模块来实现。

采样率为 DataClock 除以积分时间再加上它进行结果计算所需的时间 CalcTime。 积分时间是 DualADC 对输入信号进行采样的时间。

Equation 9

$$SampleRate = \frac{DataClock}{2^{Bits+2} + CalcTime}$$

它计算结果所耗费的时间 CalcTime 与 CPU 时钟成反比。 CalcTime 必须设置为大于计算结果所需的值。最小 CalcTime 等于 260 个 CPU 周期，且必须以 DataClock 的形式表示。 CalcTime 还可以增加到最小值之上以优化采样率。

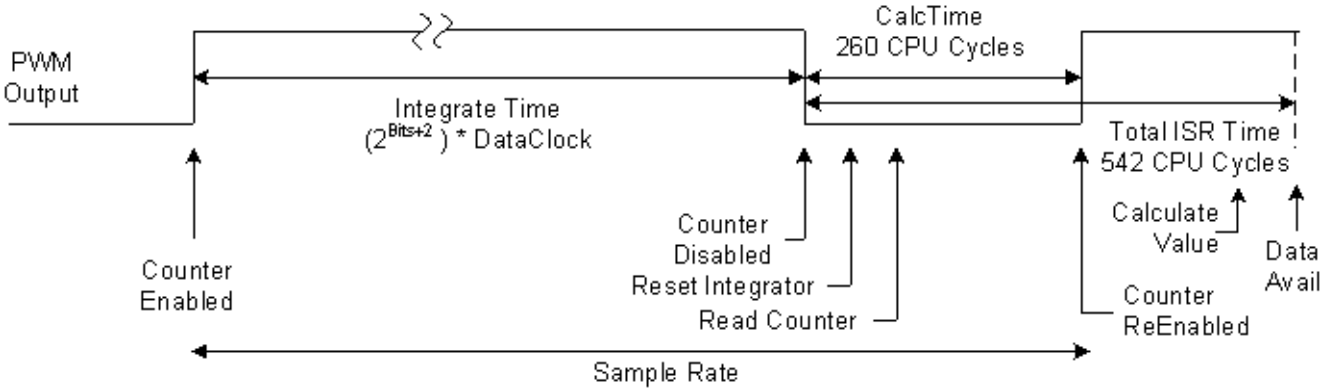
Note 2^{Bits+2} 与 CalcTime 的和不得超过 $2^{16}-1$ 或 65,535。

Equation 10

$$CalcTime \geq \frac{DataClock * 260}{CPUClock}$$

16-bit PWM 编程为输出高电平信号，该信号为 2^{Bits+2} 乘以 DataClock。 例如，如果分辨率设置为 10 位，则在 4096 (2^{10+2}) 个 DataClock 周期内，PWM 输出保持高电平。 在 PWM 执行最小结果计算和复位积分器所需的时间内，PWM 输出将保持低电平。 此周期由 CalcTime 参数控制。 还可以调整 CalcTime 以帮助提供更准确的采样率和 DataClock。 PWM 总周期是积分时间与 CalcTime 之和。

Figure 3. 与 PWM 输出相关的 DualADC 定时



当启动第一个读数时，将计算 PWM 配置，对积分器进行复位，并且计数器都复位为 FFh。 初始延迟始终至少等于计算的延迟时间。 PWM 仅在第一次读取之前初始化。 设置一次比较寄存器和周期寄存器后，它们不必重新初始化，除非更改分辨率或计算时间。 当 PWM 计数小于或等于积分值时，输出变为高电平，使得 8-bit 计数器进行倒计数。 PWM 输出保持高电平，直到计数器达到零。 此时，禁用 8-bit 计数器的时钟，并生成 PWM 中断。

8-bit 软件计数器的初始值设置为最大负数的 $2^{Bits}/64$ 倍。 每当 8-bit 计数器溢出时，将执行 8-bit 计数器的中断，软件计数器增加 1。

如果 ADC 的输入大于或等于最大正值，8-bit 计数器将在 DataClock 每次正跃变时递增。 如果 ADC 的输入小于或等于最大负输入值，8-bit 计数器将永远不再递减，因此不再生成中断。 理想情况下接近模拟接地的输入将允许计数器递增一半时间。 根据输入电压电平，8-bit 计数器中断计数的变化范围从 0 ～ $(2^{Bits+2})/256$。 例如，如果分辨率设置为 10 位，则 PWM 比较值设置为 2^{10+2} (4096)。 这意味着在积分周期内，处理器最多可以中断 4096/256 （即 16）次。

由于 DualADC 控制基于中断，对于结果，采样时间相对较长，因此在处理采样时期望处理器等待，这是不合理的。 ADC 子程序与主程序之间的主要通信是一个可轮询的标志。 如果 DualADC_bfStatus 的最高有效位非零，则 DualADC_iResultn 中提供新数据 (n=1,2)。 API 可用于检查数据标志和检索数据。

此数据处理程序设计基于轮询。 如果需要基于中断的数据处理程序，则用户可以将数据处理程序代码插入汇编文件中的中断子程序 DualADC_CNTn_ISR *DualADCINT.asm*. 插入代码的位置被清晰标记。

CPU 利用率

DualADC 需要 CPU 时间来计算结果，以及每次硬件计数器溢出时，使软件计数器递增。 CPU 开销取决于三个变量： CPU 时钟、DataClock 和输入电压。 起先，输入电压影响 ADC 的 CPU 开销似乎很奇怪。 接近或低于 -Vref 的输入电压需要很小的 CPU 开销。 接近或大于 +Vref 的输入电压需要较多的 CPU 开销。 下面的等式假设两个输入的输入信号相同。 要计算给定输入所需的 CPU 周期，请参考下列等式。

Equation 11

$$CPUcycles = PWM16\_IRQ\_CPUcycles + \left(\frac{2^{Bits}}{64} * \left(\frac{Vref + Vin}{2 * Vref}\right) * (Counter\_IRQ\_CPUcycles * 2)\right)$$

Equation 12

$$CPUcycles = 542 + \left(\frac{2^{Bits}}{64} * \left(\frac{Vref + Vin}{2 * Vref}\right) * (37 * 2)\right)$$

要根据 10 位分辨率计算最大 CPU 周期，请将 Vin 设置为 Vref，并参考下列等式。

Equation 13

$$CPUcycles = 542 + \left(\frac{2^{10}}{64} * \left(\frac{Vref + Vref}{2 * Vref}\right) * 74\right) = 542 + (16 * 1 * 74) = 1726$$

要计算 DualADC 的 CPU 利用率百分比，请参考下列等式。

Equation 14

$$Percent\_CPU\_Utilization = \frac{Sample\_Rate * CPUcycles}{CPU\_freqency} * 100$$

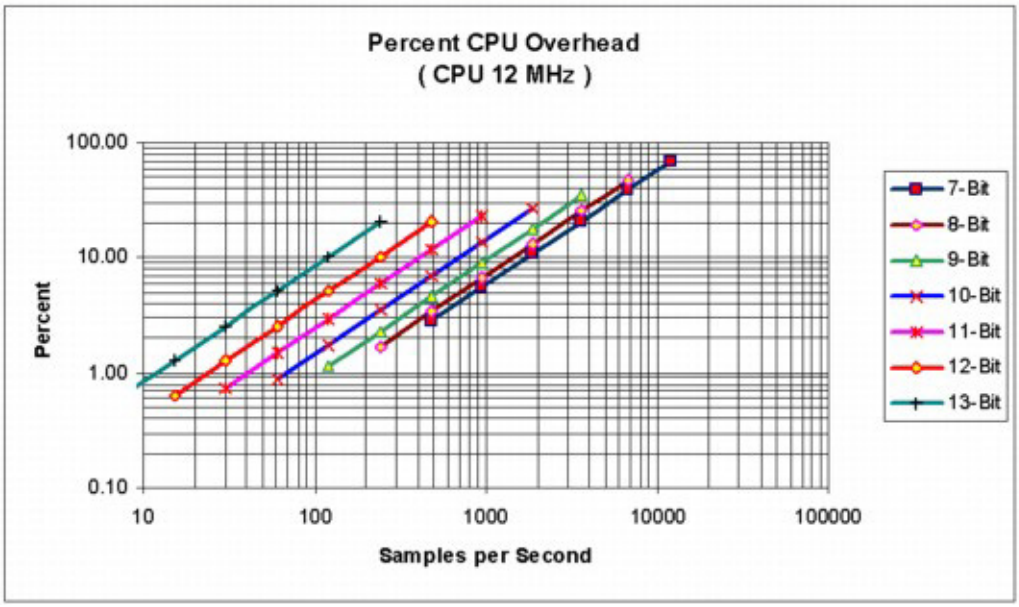
将分辨率设置为 10 位 （如上例所示），采样率设置为 1000 样品 / 秒，CPU 时钟设置为 12 MHz，则 （如下面的等式所示）利用率计算为 14.4%。

Equation 15

$$Percent\_CPU\_Utilization = \frac{1000 * 1726}{12MHz} * 100 = 14.4\%$$

下图显示了受支持采样率和分辨率的 CPU 利用率。 默认 CPU 速度设置为 12 MHz。

Figure 4. CPU 使用与采样率



通道之间的差异

如果使用 DualADC，当测量同一输入电压时，通道之间会有一些差异。 此差异的产生原因是因为开关电容模块放大器和列 AGND 缓冲区中的输入偏移差异。 通过将相同信号布线电容到每个 ADC 通道，可以轻松补偿此通道间偏差。 可以将其中一个通道作为参考，在每次读数之后将扣除后面通道之间的差异。

频率抑制

通过选择适当的积分时间，可以抑制特定噪声源。 要抑制噪声源及其谐波，请选择等于噪声信号积分周期的积分时间。 如果抑制多个信号，请选择等于全部两个信号的积分周期的积分时间。

例如，如果抑制 50 Hz 和 60 Hz 信号导致的噪声，请选择包含 50 Hz 和 60 Hz 信号的积分时间的周期。

Equation 16

$$IntegrateTime = 6 * \frac{1}{60} = 5 * \frac{1}{50} = 100mSec$$

100 ms 的 IntegrateTime 将同时抑制 50 Hz 和 60 Hz 以及这些信号的任何谐波。 接下来，计算生成适当的 IntegrateTime 所需的 DataClock。

Equation 17

$$DataClock = \frac{2^{Bits+2}}{IntegrateTime}$$

请注意，虽然 CalcTime 影响采样率，但是此计算中不使用它。 IntegrateTime 是 DualADC 实际对输入电压进行采样时的周期。 采样率基于 IntegrateTime 和它计算结果所花费的时间。

示例

在给定应用场合下，需要 100 ms 的 IntegrateTime 和 13 位的 A/D 分辨率。对于 100 ms 的 IntegrateTime，数据时钟必须如下所示。

Equation 18

$$DataClock = \frac{2^{Bits+2}}{IntegrateTime} = \frac{2^{13+2}}{100ms} = 327.7kHz$$

对于 Data Clock，CalcTime 必须根据 DataClock 和 CPU 时钟计算。如果 CPU 时钟为 12 MHz，则最少计算时间如下。

Equation 19

$$CalcTime \geq \frac{260 \times DataClock}{CPU\_Clock}$$

此 CalcTime 应当四舍五入到最近的整数值，在此示例中为 8。要确定采样率，请按如下继续操作。

Equation 20

$$SampleRate = \frac{DataClock}{2^{Bits+2} + CalcTime}$$

如果需要较长的采样率，则可以增加 CalcTime 直到 CalcTime + 2^{13+2} 小于或等于 2^{16} - 1 (65535)。

直流和交流电气特性

除非下表中另外指定，否则：T_A = 25° C，Vdd = 5.0V，功耗 = 高，运算放大器偏压 = 低，输出参考在 P2[6] 上 1.25 外部 Vref 的情况下在 P2[4] 上的 2.5V 外部模拟接地，分辨率设置为 13 位。

Table 1. CY8C29/27/24xxx, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52 系列 PSoC 器件的 5.0V DualADC 直流和交流电气特征

参数	典型值	限制	单位	条件和注释
输入				
输入电压范围	---	Vss 到 Vdd		Ref Mux = Vdd/2 ± Vdd/2
输入电容[1]	3	---	pF	
输入电阻	1/(C*clk)	---	Ω	
分辨率	---	7 到 13	比特	
采样率	---	4 到 10,000	SPS	
信噪比	77	---	dB	
直流精度				
DNL	2	---	LSB	列时钟 2 MHz
INL	1.0	---	LSB	
偏移误差	9	---	mV	

参数	典型值	限制	单位	条件和注释
增益误差				
包括参考增益误差	3.0	--	% FSR	
不包括增益误差[2]	0.1	--	% FSR	
工作电流				
低功耗	370	---	µA	
中等功耗	1200	---	µA	
高功耗	4000	---	µA	
数据时钟	---	0.125 到 8.0	MHz	数字模块和模拟列时钟的输入

The following values are indicative of expected performance and based on initial characterization data. 除非下表中另外指定，否则所有限制保证：T_A = 25° C，Vdd = 3.3V，功耗 = 高，运算放大器偏压 = 低，输出参考在 P2[6] 上 1.25 外部 Vref 的情况下在 P2[4] 上的 1.64V 外部模拟接地，分辨率设置为 13 位。

Table 2. CY8C29/27/24xxx, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52 系列 PSoC 器件的 3.3V DualADC 直流和交流电气特征

参数	典型值	限制	单位	条件和注释
输入				
输入电压范围	---	Vss 到 Vdd		Ref Mux = Vdd/2 ± Vdd/2
输入电容[1]	3	---	pF	
输入电阻	1/(C*clk)	---	Ω	
分辨率	---	7 到 13	比特	
采样率	---	4 到 10,000	SPS	
信噪比	77	---	dB	
直流精度				
DNL	2	---	LSB	列时钟 2 MHz
INL	1.0	---	LSB	
偏移误差	4	---	mV	
增益误差				
包括参考增益误差	3.0	--	% FSR	
不包括增益误差[2]	0.4	--	% FSR	
工作电流				

参数	典型值	限制	单位	条件和注释
低功耗	300	---	µA	
中等功耗	1000	---	µA	
高功耗	3800	---	µA	
数据时钟	---	0.125 到 8.0	MHz	数字模块和模拟列时钟的输入

电气特性注释

1. 包括 I/O 引脚。
2. 参考增益误差测量方法是将 $V_{RefHigh}$ 外部参考与通过测试复用器并返回至引脚的 V_{RefLow} 进行比较。

除非下表中另外指定，否则所有限制保证：T_A = -40° C 到 +85° C，Vdd = 5.0V ± 10%，功耗 = 高，运算放大器偏压 = 低，输出参考在 P2[6] 上 1.25 外部 Vref 的情况下在 P2[4] 上的 2.5V 外部模拟接地，分辨率设置为 12 位。

Table 3. CY8C26/25xxx 系列 PSoC 器件的 5.0V DualADC 直流和交流电气特征

参数	典型值[1]	限制	单位	条件和注释
输入				
输入电压范围[2]	---	Vss 到 Vdd		Ref Mux = Vdd/2 ± Vdd/2
输入电容[3]	0.8	---	pF	
输入电阻[4,5]	1/(C*clk)	---	Ω	
分辨率	---	7 到 13	比特	2 的补码
采样率	---	4 到 10,000[6]	SPS	每秒采样数 (sps)
信噪比[7]	68		dB	在 100 sps 条件下
直流精度				
INL	0.5	1	LSB	
DNL	0.25	0.5	LSB	
偏移误差	12	49	mV	使用外部 AGND
增益误差	0.5	1.5	% FSR	相对于参考输出
工作电流				
低功耗	185	---	µA	
中等功耗	450	---	µA	
高功耗	1280	---	µA	
数据时钟	---	0.032 到 8.0[6]	MHz	数字模块和模拟列时钟的输入

除非下表中另外指定，否则所有限制保证：T^A = -40° C ～ +85° C，Vdd = 3.0 ～ 3.6V，功耗 = 高，运算放大器偏压 = 低，输出参考在 P2[6] 上 1.25 外部 Vref 的情况下在 P2[4] 上的 1.64V 外部模拟接地，分辨率设置为 12 位。

Table 4. CY8C26/25xxx 系列 PSoC 器件的 3.3V DualADC 直流和交流电气特征

参数	典型值[1]	限制	单位	条件和注释
输入				
输入电压范围[2]	---	Vss +/- Vdd		Ref Mux = Vdd/2 ± Vdd/2
输入电容[3]	0.8	---	pF	
输入电阻[4,5]	1/(C*clk)	---	Ω	
分辨率	---	7 到 13	比特	2 的补码
采样率	---	4 到 10,000[6]	KSPS	每秒采样数 (sps)
信噪比[7]	65		dB	在 100 sps 条件下
直流精度				
INL	0.5	1	LSB	
DNL	0.25	0.5	LSB	
偏移误差	12	49	mV	
增益误差	0.5	2.5	% FSR	相对于参考输出
工作电流				
低功耗	150	---	µA	
中等功耗	350	---	µA	
高功耗	1120	---	µA	
数据时钟	---	0.125 到 8[6]	MHz	数字模块和模拟列时钟的输入

电气特性注释

1. 典型值表示 +25° C 时的参数标准。
2. 高于最大值的输入电压将生成最大正读数。 低于最小值的输入电压将生成最大负读数。
3. 仅限用户模块，不包括 I/O 引脚。
4. 输入电容或电阻仅在模拟模块的输入直接连接到引脚时适用。
5. C = 输入电容，clk = 数据时钟 （模拟列时钟）。
6. 除非另有指定，否则规范针对的是采样率 100 sps 和最大数据时钟 8 MHz。 采样率取决于数据时钟和分辨率。
7. 信噪比 = 全量程单一音频除以 Fsample/2 积分的总噪声的功率比。

放置

ADC （开关电容）模块可以放置在任意开关电容 PSoC 模块中。 它们必须每个都能单独驱动其所在特定列的比较器总线。 换句话说，两个模块中的每个模块必须位于不同的列中，不能与连接到比较器总线的其他开关电容模块共享一列。

计数器模块可以放置在任何可用数字模块中，但是 PWM16 只能放在特定位置。 在 CY8C27xxx 器件系列中，PWB16 (LSB/MSB) 的可能放置有 DBB00/DBB01、DBB01/DCB02、DBB10/DBB11 和 DBB11/DCB12。 在 CY8C29/24/22xxx 器件系列中，PWM16 可以放置在任何两个连续的数字模块中。 在 CY8C26/25xxx 系列中，PWB16 (LSB/MSB) 的可能放置有 DBA01/DBA02 和 DCA05/DCA06。

两个计数器模块和每个 PWM 模块都有中断服务子程序。 计数器模块需要具有比 PWM16 模块高的中断优先级。 因此，建议将计数器模块放置在比 PWM16 模块顺序低的模块位置。

Note 当初始选择 DualADC 时，将显示一条警告，指出 “Resource allocation prevents placement”（资源分配阻止放置）。 如果原始放置在同一列中有两个 ADC 模块，则显示此条警告。 只需将每个 ADC 块移动到其自己的列。

参数和资源

ADC Input1、ADC Input2

ADC 输入的选择是在放置模拟 PSoC 模块之后完成的。 八个开关电容模块具有不同的输入选择。 每个模块都可以连接到其大部分相邻模块，而某些模块则可以直接连接到外部输入引脚。 放置模拟模块时必须考虑如何获取其输入信号。 某些放置允许直接从封装引脚将输入路由至输入。 这些直接连接允许精确测量供电轨 40 mV 范围内的输入。 信号还可以通过列复用器之一、CT 模块测试复用器之一路由，以及路由至 DualADC 还可以测量电源轨附近的信号的模拟列上。 对于两个 ADC 输入的每个输入，都提供了一个选择。

ClockPhase1、ClockPhase2

选择时钟相位是为了将一个开关电容模拟 PSoC 模块的输出与另一个模块的输入同步。开关电容模拟 PSoC 模块使用两相时钟 （Φ_1、Φ_2）获取和传输信号。 通常，DualADC 的输入是在 Φ_1 中采样的，这是 “ 正常 ” 设置。 这便产生一个问题：许多用户模块在 Φ_1 期间将其输出自动归零，仅在 Φ_2 期间提供有效输出。 如果此类模块的输出馈送到 DualADC 的输入，则 DualADC 获取自动归零的输出，而不是有效信号。 时钟相位选择允许交换相位，因此输入信号现在是在 Φ_2 期间获取的，这是 “ 交换 ” 设置。 针对两个开关电容模块中的每一个模块，都提供了一个选择。

时钟和积分器列时钟

DataClock 确定采样率和信号采样窗口。 此时钟必须路由到计数器模块、16-bit PWM 模块和包含积分器列的列时钟的时钟输入。

Note 必须将积分器开关电容模块的列时钟手动设置为 SAME 时钟。 所有六个模块必须使用同一时钟，否则 DualADC 用户模块将无法正常工作。

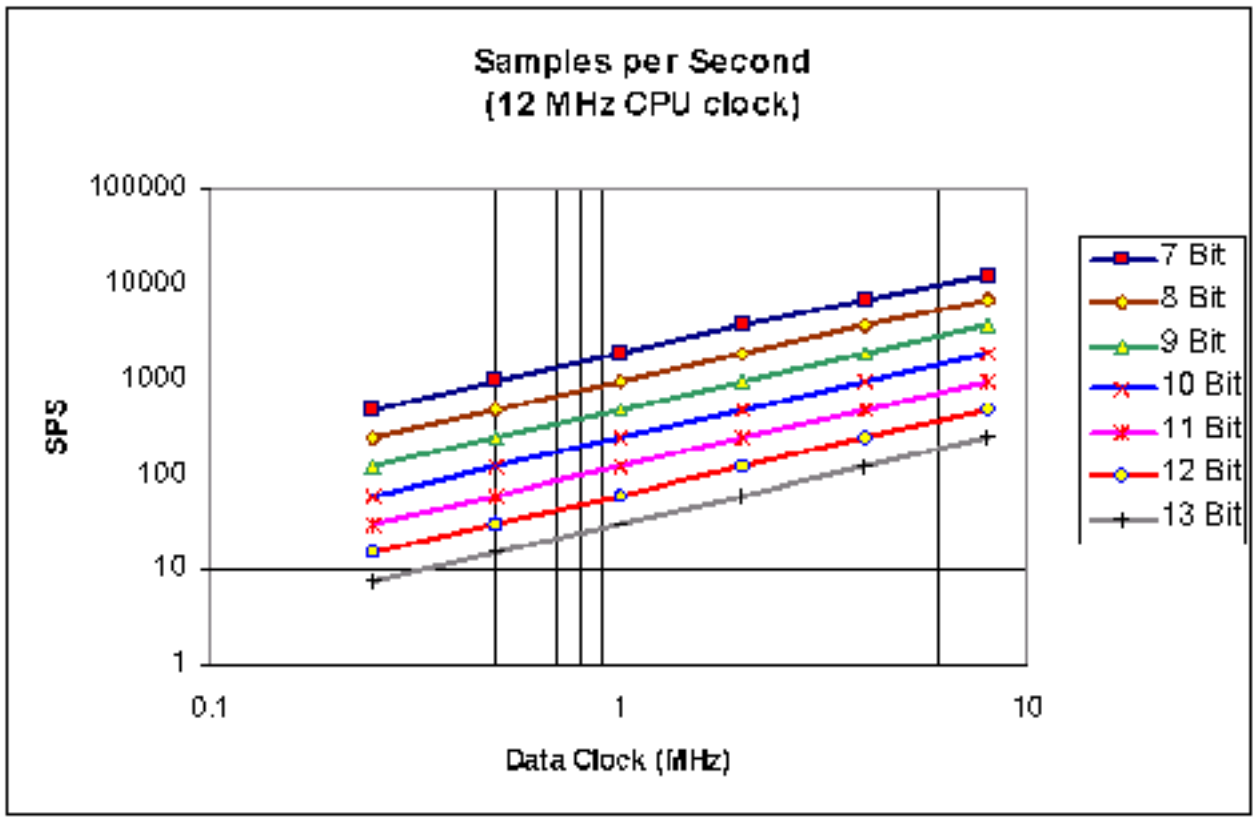
此参数设置仅设置计数器模块和 PWM 模块的时钟。 此时钟可以是时钟速率介于 125 kHz 和 8 MHz 之间的任何源。

Equation 21

$$SampleRate = \frac{DataClock}{2^{Bits+2} + CalcTime}$$

下图显示了 DualADC 的每个分辨率选项的可能采样率。

Figure 5. 每秒采样数与数据时钟



ADCResolution

ADCResolution 选择允许在器件编辑器中设置 DualADC 的分辨率。 虽然有一个用于设置或更改分辨率的 API 子程序，但是如果在器件编辑器中进行了设置，则不需要使用该子程序。 还可以随时使用 API 调用来更改分辨率，但是，要停止 DualADC 且必须重新启动它。 有效分辨率设置为 7 到 13（含）。

CalcTime

CalcTime 是下一积分周期前 CPU 用于计算中间积分结果所耗费的时间。 计算 “CalcTime” 结果所耗费的时间与 CPU 时钟成反比变化。此值必须是数据时钟的形式。最小 CPU 计算时间为 260 个 CPU 时钟。 还可以增加 CalcTime 以优化采样率。

Note 一定要注意确保 CalcTime + 2^{Bits+2} 不超过 $2^{16}-1$ 或 65,535。

下面是一个用于确定 CalcTime 设置的等式。

$$CalcTime \geq \frac{DataClock * 260}{CPUClock}$$

下表显示了可以为 CalcTime 参数选择的范围。 使用上述等式可设置给定应用场合的可用范围的下限。

Table 5. CalcTime 范围

分辨率	积分时间 （DataClock 计数）	CalcTime 范围 （DataClock 计数）
7	512	1 到 65,023
8	1,024	1 到 64,511
9	2,048	1 到 63,487
10	4,096	1 到 61,439
11	8,192	1 到 57,343
12	16,384	1 到 49,151
13	32,768	1 到 32,767

例如，如果 DataClock 设置为 1.5 MHz，CPU 在 12 MHz 运行，则 CalcTime 应当设置为大于或等于 33 （请参见下面的等式）。

Equation 22

$$CalcTime \geq \frac{DataClock * 260}{CPUClock}$$

DataFormat

此选择确定返回结果的格式。 如果选择 “ 有符号 ” 且 “N” 是选定的分辨率，则结果范围介于 -2^{N-1} 到 $2^{N-1}-1$。 如果选择 “ 无符号 ”，则结果将介于 0 和 $2^{N}-1$ 之间。 有关每种数据格式的结果范围和分辨率，请参见下表。

Table 6. 数据格式结果范围

分辨率设置	有符号数据格式	无符号数据格式
7	-64 到 63	0 到 127
8	-128 到 127	0 到 255
9	-256 到 255	0 到 511
10	-512 到 511	0 到 1023
11	-1024 到 1023	0 到 2047
12	-2048 到 2047	0 到 4095
13	-4096 到 4095	0 到 8191

参考复用器全局资源

处理模拟到数字转换器 (ADC) 时，最重要的全局资源为 RefMux。 RefMux 的设置确定 ADC 的可用输入电压范围。 下表显示了 Vdd 为 5 和 3.3 伏的范围。

Table 7. 与 RefMux 设置有关的 CY8C26/25xxx 输入电压范围

RefMux 设置	Vdd = 5 伏	Vdd = 3.3 伏
(Vdd/2) ± 带隙	$1.2 < V_{in} < 3.8$	$0.35 < V_{in} < 2.95$
(Vdd/2) ± (Vdd/2)	$0 < V_{in} < 5$	$0 < V_{in} < 3.3$
(2* 带隙) ± 带隙	$1.3 < V_{in} < 3.9$	NA
(2* 带隙) ± P2[6]	$(2.6 - V_{P2[6]}) < V_{in} < (2.6 + V_{P2[6]})$	NA
P2[4] ± 带隙	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$
P2[4] ± P2[6]	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$

Table 8. 每个参考复用器设置的 CY8C29/27/24xxx, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52 输入电压范围

RefMux 设置	Vdd = 5 伏	Vdd = 3.3 伏
(Vdd/2) ± 带隙	$1.2 < V_{in} < 3.8$	$0.35 < V_{in} < 2.95$
(Vdd/2) ± (Vdd/2)	$0 < V_{in} < 5$	$0 < V_{in} < 3.3$
带隙 ± 带隙	$0 < V_{in} < 2.6$	$0 < V_{in} < 2.6$
(1.6* 带隙) ± (1.6* 带隙)	$0 < V_{in} < 4.16$	NA
(2* 带隙) ± 带隙	$1.3 < V_{in} < 3.9$	NA
(2* 带隙) ± P2[6]	$(2.6 - V_{P2[6]}) < V_{in} < (2.6 + V_{P2[6]})$	NA
P2[4] ± 带隙	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$
P2[4] ± P2[6]	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$

中断生成控制

下列参数可用的情况仅限于 **启用中断生成控制** 复选框时，会有一个附加参数变为可用。 可以通过依次单击以下菜单获得此参数: **项目 > 设置 > 芯片编辑器** 当外覆层的多个用户模块所共享的中断用于多个外覆层时，中断生成控制非常重要。

IntDispatchMode

IntDispatchMode 参数用于指定如何为位于同一模块但在不同外覆层中的多个用户模块共享的中断来处理中断请求。 选择 “ActiveStatus” 会导致固件测试在为共享中断请求提供服务之前哪个外覆层处于活动状态。 每当请求共享中断时，都会发生此测试。 这会增加延迟，还会生成为共享中断请求

提供服务的不确定过程，但是不需要任何 RAM。 选择 “OffsetPreCalc” 会导致固件仅当初始加载外覆层时计算共享中断请求的源。 此计算减少了中断延迟，并生成为共享中断请求提供服务的确定过程，但是耗费了一字节 RAM。

应用程序编程接口

应用程序编程接口 (API) 子程序作为用户模块的一部分提供，从而使设计人员能够采用更高级的方式处理模块。 本节指定每个函数的接口，以及 “ 引用 ” 文件所提供的相关常量。

Note 在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可能通过调用 API 函数发生更改。 如果在调用后需要 A 和 X 的值，则调用函数负责在调用前保留 A 和 X 的值。 选择此 “ 寄存器易失 ” 策略是为了提高效率，自 PSoC Designer 1.0 版起已强制使用此策略。 C 编译器自动遵循此要求。 汇编语言程序员也必须确保其代码遵守这一策略。 虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们将来也会如此。

API 子程序可以初始化、配置、开始采样、停止和读取 ADC 生成的数据。

DUALADC_Start

说明：

针对此用户模块执行所有必需的初始化，设置开关电容 PSoC 模块的功耗水平。

C 原型：

```
void  DualADC_Start (BYTE bPowerSetting)
```

汇编：

```
mov   A, DualADC_HIGHPOWER
lcall  DualADC_Start
```

参数：

PowerSetting： 一个用于指定功耗水平的字节。 在复位和配置之后，分配给 DualADC 的模拟 PSoC 关闭电源。 下表给出了在 C 语言程序和汇编语言程序中提供的符号名称及其相关数值。

符号名称	值
DualADC_OFF	0
DualADC_LOWPOWER	1
DualADC_MEDPOWER	2
DualADC_HIGHPOWER	3

功耗水平对模拟性能有影响。 正确的功耗设置与数据时钟的采样率密切相关，必须为每个应用场合确定正确的功耗设置。 建议开始设计开发时选择全功率。 可以稍后进行测试，以确定可以将功耗设置设置到多低的程度。

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页指针寄存器也会出现这种情况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。 目前，仅修改 CUR_PP 页指针寄存器。

DUALADC_Stop

说明：

将开关电容积分器模块的功耗水平设置为 Off。 当未使用 DualADC 且用户想要省电时，进行此设置。 此子程序关闭模拟开关电容模块的电源，禁用数字模块。 要实现最低功耗水平，还应当从数字模块中删除时钟。

C 原型：

```
void  DualADC_Stop()
```

汇编：

```
lcall  DualADC_Stop
```

参数：

无

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页指针寄存器也会出现这种情况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。

DUALADC_SetPower

说明：

设置开关电容 PSoC 模块的功耗水平。

C 原型：

```
void  DualADC_SetPower (BYTE bPowerSetting)
```

汇编：

```
mov   A, [bPowerSetting]
lcall  DualADC_SetPower
```

参数：

PowerSetting 与用于启动 API 子程序的 PowerSetting 参数相同。 允许用户在运行 ADC 时更改功耗水平。

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页指针寄存器也会出现这种情况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。 目前，仅修改 CUR_PP 页指针寄存器。

DUALADC_SetResolution

说明：

设置 A/D 转换器的分辨率。

C 原型：

```
void  DualADC_SetResolution (BYTE bResolution)
```

汇编：

```
mov   A, [bResolution]
lcall  DualADC_SetResolution
```

参数：

分辨率： 可以在器件编辑器或用户固件中设置 A/D 转换器的分辨率。 如果未在固件中设置，则默认情况下 ADC 将使用器件编辑器中的分辨率设置。 分辨率的值可以设置为 7 到 13 位。

返回值：

无

副作用：

停止 A/D 转换器。 A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页指针寄存器也会出现这种情况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。 目前，仅修改 CUR_PP 页指针寄存器。

DUALADC_GetSamples

说明：

初始化并启动 ADC 算法以收集指定的采样数。 请记住通过调用 *M8C.inc* 或 *M8C.h* 中定义的 M8C_EnableGInt 宏调用，启用全局中断。

C 原型：

```
void  DualADC_GetSamples (BYTE bNumSamples)
```

汇编：

```
mov   A, [bNumSamples]
lcall  DualADC_GetSamples
```

参数：

bNumSamples： 8-bit 值，用于设置要检索的采样数。 为值 “0” 会导致 ADC 连续运行。

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页指针寄存器也会出现这种情况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。 目前，仅修改 CUR_PP 页指针寄存器。

DUALADC_StopAD

说明：

立即停止 ADC。

C 原型：

```
void  DualADC_StopAD()
```

汇编：

```
lcall  DualADC_StopAD
```

参数：

无

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页指针寄存器也会出现这种情况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。

DUALADC_fIsDataAvailable

说明：

当数据转换完成且数据可用于读取时，返回非零值。

C 原型：

```
BYTE  DualADC_fIsDataAvailable()
```

汇编：

```
lcall  DualADC_fIsDataAvailable
```

参数：

无

返回值：

当数据可用时返回非零值。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页指针寄存器也会出现这种情况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。 目前，仅修改 CUR_PP 页指针寄存器。

DUALADC_iGetData1

说明：

返回上次为 ADC Input1 转换的数据。应当在获取数据之前调用 DUALADC_fIsDataAvailable()，以确保数据有效。 必须在下次转换周期完成之前检索数据，否则数据将被覆盖。 如果对此函数的调用正好是在积分周期结束时完成的，则返回的数据有可能损坏。 因此，极力建议以高于采样率的频率进行数据检索，或者如果不能保证在调用此函数之前关闭中断，则应进行数据检索。

C 原型：

```
INT   DualADC_iGetData1()
```

汇编：

```
lcall  DualADC_iGetData1
```

参数：

无

返回值：

返回转换的整数值。 在汇编程序中，MSB 返回到 X 寄存器中，LSB 返回到累加器中。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页指针寄存器也会出现这种情况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。 目前，仅修改 CUR_PP 页指针寄存器。

DUALADC_iGetData2

说明：

返回上次为 ADC Input2 转换的数据。应当在获取数据之前调用 DUALADC_fIsDataAvailable()，以确保数据有效。 必须在下次转换周期完成之前检索数据，否则数据将被覆盖。 如果对此函数的调用正好是在积分周期结束时完成的，则返回的数据有可能损坏。 因此，极力建议以高于采样率的频率进行数据检索，或者如果不能保证在调用此函数之前关闭中断，则应进行数据检索。

C 原型：

```
INT   DualADC_iGetData2()
```

汇编：

```
lcall  DualADC_iGetData2
```

参数：

无

返回值：

返回转换的整数值。 在汇编程序中，MSB 返回到 X 寄存器中，LSB 返回到累加器中。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页指针寄存器也会出现这种情况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。 目前，仅修改 CUR_PP 页指针寄存器。

DUALADC_ClearFlag

说明：

清除数据可用标志。

C 原型：

```
void  DualADC_ClearFlag()
```

汇编：

```
lcall  DualADC_ClearFlag
```

参数

无

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页指针寄存器也会出现这种情况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。 目前，仅修改 CUR_PP 页指针寄存器。

DUALADC_iGetData1ClearFlag

说明：

返回上次为 ADC Input1 转换的数据，并清除数据可用标志。应当在获取数据之前调用 DUALADC_fIsDataAvailable()，以确保数据有效。 必须在下次转换周期完成之前检索数据，否则数据将被覆盖。 如果对此函数的调用正好是在积分周期结束时完成的，则返回的数据有可能损坏。 因此，极力建议以高于采样率的频率进行数据检索，或者如果不能保证在调用此函数之前关闭中断，则应进行数据检索。

C 原型：

```
INT   DualADC_iGetData1ClearFlag()
```

汇编：

```
lcall  DualADC_iGetData1ClearFlag
```

参数：

无

返回值：

返回转换的整数值。 在汇编程序中，MSB 返回到 X 寄存器中，LSB 返回到累加器中。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页指针寄存器也会出现这种情况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。 目前，仅修改 CUR_PP 页指针寄存器。

DUALADC_iGetData2ClearFlag

说明：

返回上次为 ADC Input2 转换的数据，并清除数据可用标志。应当在获取数据之前调用 DUALADC_fIsDataAvailable()，以确保数据有效。 必须在下次转换周期完成之前检索数据，否则数据将被覆盖。 如果对此函数的调用正好是在积分周期结束时完成的，则返回的数据有可能损坏。 因此，极力建议以高于采样率的频率进行数据检索，或者如果不能保证在调用此函数之前关闭中断，则应进行数据检索。

C 原型：

```
INT   DualADC_iGetData2ClearFlag()
```

汇编：

```
lcall  DualADC_iGetData2ClearFlag
```

参数：

无

返回值：

返回转换的整数值。 在汇编程序中，MSB 返回到 X 寄存器中，LSB 返回到累加器中。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页指针寄存器也会出现这种情况。 如果需要，由调用函数负责将调用前后的数值保存在 fastcall16 函数内。 目前，仅修改 CUR_PP 页指针寄存器。

Note 函数 DUALADC_ClearFlag、DUALADC_iGetData1ClearFlag 和 DUALADC_iGetData2ClearFlag 都清除相同的标志。 提供它们可在清除转换完成标志时提供最大的灵活性。 当 A/D 转换完成时，用户可以选择忽略一个或全部两个通道的结果，并且可以只清除标志而不检索数据。

固件源代码示例

下面是一个以汇编代码编写的项目示例。

```
;;; Sample ASM Code for the DualADC
;;;
;;; Continuously sample using the DualADC and store the values in RAM.
;;;

include "m8c.inc"       ; part specific constants and macros
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

;; Create storage for readings
area bss(RAM)
iResult1:       BLK   2  ; ADC1 result storage
iResult2:       BLK   2  ; ADC2 result storage
export iResult1          ; Export results in case they are
export iResult2          ; used elsewhere.

area text(ROM,REL)
export _main

_main:

    M8C_EnableGInt                ; Enable interrupts
    mov   A, 10                   ; Set resolution to 10 Bits
    call  DUALADC_SetResolution

    mov   A, DUALADC_HIGHPOWER    ; Set Power and Enable A/D
    call  DUALADC_Start

    mov   A, 00h                  ; Start A/D in continuous sampling mode
    call  DUALADC_GetSamples


;A/D conversion loop
loop1:

wait:                             ; Poll until data is complete
    call  DUALADC_fIsDataAvailable
    jz    wait
    call  DUALADC_ClearFlag       ; Reset flag

    call  DUALADC_iGetData1       ; Get ADC1 Data (X=MSB A=LSB)
    mov   [iResult1+1],A          ; Store LSB
    mov   [iResult1+0],X          ; Store MSB

    call  DUALADC_iGetData2       ; Get ADC2 Data (X=MSB A=LSB)
    mov   [iResult2+1],A          ; Store LSB
    mov   [iResult2+0],X          ; Store MSB
    jmp   loop1
```

随后是以 C 语言编写的项目示例。

```
//-------------------------------------------------------------------------
//  Sample C Code for the DualADC
//  Continuously Sample and call a user function with the data.
//  This example differs from the ASM example, in that the DataAvailable
//  flag is automatically cleared when the second value is read, instead of
//  clearing the flag prior to reading the data.
//
//-------------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

extern void User_Function(int iResult1, int iResult2);

void main(void)

{

   int iResult1, iResult2;

   M8C_EnableGInt;                        // Enable global interrupts
   DUALADC_Start(DUALADC_HIGHPOWER);      // Turn on Analog section
   DUALADC_SetResolution(10);             // Set resolution to 10 Bits
   DUALADC_GetSamples(0);                 // Start ADC to read continuously

   for(;;)

   {

      while(DUALADC_fIsDataAvailable() == 0);  // Wait for data to be ready
      iResult1 = DUALADC_iGetData1();          // Get Data from ADC Input1
      iResult2 = DUALADC_iGetData2ClearFlag(); // Get Data from ADC Input2
                                               // and clear data ready flag

      User_Function(iResult1,iResult2);        // User function to use data
   }

}
```

配置寄存器

这些寄存器通过初始化和 API 库进行配置。 用户不必直接更改或读取这些寄存器。 此部分仅供参考。

ADC 是开关电容 PSoC 模块。 它配置为生成模拟调制器。 要生成调制器，需要将该模块配置为带有参考反馈的积分器，该积分器将输入值转换为数字脉冲流。 输入复用器确定将对哪些信号执行数字化。

Table 9. 模块 ADC1: 寄存器 CR0

位	7	6	5	4	3	2	1	0
值	1	0	0	1	0	0	0	0

Table 10. 模块 ADC1: 寄存器 CR1

位	7	6	5	4	3	2	1	0
值	ACMux、AMux			0	0	0	0	0

当模块放置在 “A” 型模块中时，使用 ACMux。 当模块放置在 “B” 型模块中时，使用 AMux。 两个字段值都取决于用户连接输入的方式。

Table 11. 模块 ADC1: 寄存器 CR2

位	7	6	5	4	3	2	1	0
值	0	1	1	0	0	0	0	0

Table 12. 模块 ADC1: 寄存器 CR3

位	7	6	5	4	3	2	1	0
值	1	1	1	FSW0	0	0	0	0

FSW0 供 PWM 中断处理程序和各种 API 使用。 如果值为 “0”，则 ADC 成为禁用的积分器。 如果值为 “1”，则 ADC 成为启用的积分器。

Table 13. 模块 ADC2: 寄存器 CR0

位	7	6	5	4	3	2	1	0
值	1	0	0	1	0	0	0	0

Table 14. 模块 ADC2: 寄存器 CR1

位	7	6	5	4	3	2	1	0
值	ACMux、AMux			0	0	0	0	0

当模块放置在 “A” 型模块中时，使用 ACMux。 当模块放置在 “B” 型模块中时，使用 AMux。 两个字段值都取决于用户连接输入的方式。

Table 15. 模块 ADC2: 寄存器 CR2

位	7	6	5	4	3	2	1	0
值	0	1	1	0	0	0	0	0

Table 16. 模块 ADC2: 寄存器 CR3

位	7	6	5	4	3	2	1	0
值	1	1	1	FSW0	0	0	0	0

FSW0 供 TMR 中断处理程序和各种 API 使用。 如果值为 “0”，则 ADC 成为禁用的积分器。 如果值为 “1”，则 ADC 成为启用的积分器。

PWM16 是用于控制 ADC 积分时间的数字 PsoC 模块。 比较值设置为 2^{Bits+2}，周期设置为 CalcTime 加上比较值。

Table 17. 模块 PWM16_MSB: 寄存器函数

位	7	6	5	4	3	2	1	0
值	0	0	1	Compare Type	Interrupt Type	0	0	1

Compare Type 标志指示捕获比较是设置为 “equal to or less than”（等于或小于）还是设置为 “less than” （小于）。 Interrupt Type 标志指示是基于捕获事件还是基于终端条件来触发中断。 这两个参数都是在设备编辑器中设置的。

Table 18. 模块 PWM16_LSB：寄存器函数

位	7	6	5	4	3	2	1	0
值	0	0	0	Compare Type	0	0	0	1

Compare Type 标志指示比较函数是设置为 “equal to or less than”（等于或小于）还是设置为 “less than” （小于）。 此参数是在设备编辑器中设置的。

Table 19. 模块 PWM16_MSB：寄存器输入

位	7	6	5	4	3	2	1	0
值	0	0	1	1	时钟			

时钟从 16 个源之一选择输入时钟。 此参数是在设备编辑器中设置的。

Table 20. 模块 PWM16_LSB：寄存器输入

位	7	6	5	4	3	2	1	0
值	启用				时钟			

设置为 “Enable” 则从 16 个源之一选择数据输入，设置为 “Clock” 则从 16 个源之一选择时钟输入。 这两个参数都是在设备编辑器中设置的。

Table 21. 模块 PWM16_MSB：寄存器输出

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	Output Enable	Output Sel	

Output Enable 标志表示启用输出。 Output Sel 标志指示 PWM16 的输出将路由到何处。 这两个参数都是在设备编辑器中设置的。

Table 22. 模块 PWM16_LSB：寄存器输出

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 23. 模块 PWM16_MSB：计数寄存器 DR0

位	7	6	5	4	3	2	1	0
值	Count(MSB)							

该 Count 是 PWM16 MSB 下降 PWM。 可以使用 PWM16 API 读取它。

Table 24. 模块 PWM16_LSB：计数寄存器 DR0

位	7	6	5	4	3	2	1	0
值	Count(LSB)							

Count 是用于停止 PWM 的 PWM16 LSB （最低有效位）。 可以使用 PWM16 API 读取它。

Table 25. 模块 PWM16_MSB：周期寄存器 DR1

位	7	6	5	4	3	2	1	0
值	Period(MSB)							

Period 用于保存依据启用或终端计数条件加载到计数器寄存器中的周期值的 MSB。 可以用设备编辑器和 PWM16 API 来对其进行设置。

Table 26. 模块 PWM16_LSB: 周期寄存器 DR1

位	7	6	5	4	3	2	1	0
值	Period(LSB)							

Period 用于保存依据启用或终端计数条件加载到计数器寄存器中的周期值的 LSB。 可以用设备编辑器和 PWM16 API 来对其进行设置。

Table 27. 模块 PWM16_MSB: 脉宽寄存器 DR2

位	7	6	5	4	3	2	1	0
值	Pulse Width(MSB)							

PulseWidth 保存用于生成比较事件的脉宽值的 MSB。 可以用设备编辑器和 PWM16 API 来对其进行设置。

Table 28. 模块 PWM16_LSB: 脉宽寄存器 DR2

位	7	6	5	4	3	2	1	0
值	Pulse Width(LSB)							

PulseWidth 保存用于生成比较事件的脉宽值的 LSB。 可以用设备编辑器和 PWM16 API 来对其进行设置。

Table 29. 模块 PWM16_MSB: 控制寄存器 CR0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	Start/ Stop(0)

Start/Stop 由 LSB 控制寄存器值控制，设置为零。

Table 30. 模块 PWM16_LSB: 控制寄存器 CR0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	Start/ Stop

设置 Start/Stop 表示启用 PWM16。 可以使用 PWM16 API 修改它

CNT 是配置为计数器的数字 PSoC 模块。 当 DR0 中的值倒数到终端计数时，将调用中断以降低软件计数器和 CNT 从 DR1 重新加载的较高值。 数据通过 DR2 输出。

Table 31. 模块 CNT1: 寄存器函数

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

Table 32. 模块 CNT1: 寄存器输入

位	7	6	5	4	3	2	1	0
值	Data				时钟			

Data 用于选择放置了 ADC 模块的列比较器。 Clock 用于从 16 个源之一选择输入时钟，它是在设备编辑器中设置的。

Table 33. 模块 CNT1: 寄存器输出

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 34. 模块 CNT1: 寄存器 DR0

位	7	6	5	4	3	2	1	0
值	Count Value							

Table 35. 模块 CNT1: 寄存器 DR1

位	7	6	5	4	3	2	1	0
值	1	1	1	1	1	1	1	1

Table 36. 模块 CNT1: 寄存器 DR2

位	7	6	5	4	3	2	1	0
值	Data Out							

Data Out 是 API 用来获取计数器值的寄存器。

Table 37. 模块 CNT1: 寄存器 CR0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	启用

Enable 用来启用 CNT。 它由 DualADC API 修改和控制

Table 38. 模块 CNT2: 寄存器函数

位	7	6	5	4	3	2	1	0
值	0	0	1	0	0	0	0	1

Table 39. 模块 CNT2: 寄存器输入

位	7	6	5	4	3	2	1	0
值	Data				时钟			

Data 用于选择放置了 ADC 模块的列比较器。 Clock 用于从 16 个源之一选择输入时钟，它是在设备编辑器中设置的。

Table 40. 模块 CNT2: 寄存器输出

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	0

Table 41. 模块 CNT2: 寄存器 DR0

位	7	6	5	4	3	2	1	0
值	Count Value							

Table 42. 模块 CNT2: 寄存器 DR1

位	7	6	5	4	3	2	1	0
值	1	1	1	1	1	1	1	1

Table 43. 模块 CNT2: 寄存器 DR2

位	7	6	5	4	3	2	1	0
值	Data Out							

Data Out 是 API 用来获取计数器值的寄存器。

Table 44. 模块 CNT2: 寄存器 CR0

位	7	6	5	4	3	2	1	0
值	0	0	0	0	0	0	0	启用

Enable 用来启用 CNT。 它由 DualADC API 修改和控制。

Table 45. 寄存器: INT_MSK1

位	7	6	5	4	3	2	1	0
值								

此处用于设置对应于 TMR 模块和 CNT 模块的掩码位，以启用其各自的中断。 实际的掩码值由每个模块的放置位置决定。

版本历史记录

版本	创作者	说明
2.2	DHA	增加了 DRC 以检查是否出现下列情况： 1. 数字和模拟资源的源时钟不同。 2. ADC 时钟大于 CPU 时钟。

Note PSoC Designer 5.1 在所有的用户模块数据手册中提供版本历史记录。 本数据表详细介绍了当前和先前用户模块版本之间的区别。